Package 'Rcell'

November 3, 2011

Type Package

Version 1.1-6

Date 2011-11-02

Title Microscopy Based Citometry Data Analysis Package

Author Alan Bush <abush@fbmc.fcen.uba.ar>

Maintainer Alan Bush <abush@fbmc.fcen.uba.ar>

Depends R (>= 2.10.0), stats, reshape, plyr (>= 1.6), ggplot2, Hmisc

Suggests hopach, chron, EBImage, xtable

Description Provides functions to load, filter and visualize microscopy based citometry data

License GPL-2

Encoding latin1

URL http://www.lbms.df.uba.ar, http://sourceforge.net/projects/cell-id

LazyData false

R topics documented:

`Rcell-package` *Cell Data Analysis and Plotting Package*

Description

Microscopy based cytometry produces hughe amount of images to be analized. Several programs can segment the acquired images and produce a dataset with the features of the found cells. This package contains functions design to analyze such datasets. It was created to analyze data from Cell-ID (`http://sourceforge.net/projects/cell-id/`), but can be extended to analyze datsets from other programs.

Details

Package:	Rcell
Type:	Package
Version:	1.1-6
Date:	2011-11-02
License:	GPL-2

Tutorials

For a introduction read the 'Getting Started with Rcell' vignette
`vignette('Rcell')`

To learn how to create complex plots read
`vignette('cplot')`

To see how to create arrays of cells images read
`vignette('cimage')`

To lear how to normalize and manipulate your data read
`vignette('transform')`

Loading Cell-ID Data in R

Once you have processed the images with Cell-ID you will have to analyse the output data. The first thing you will have to do is load your data into R.

`load.cellID.data`: this function searches a specified directory (the working directory by default) for folders that match a customizable pattern, usually PosXX where XX is the position number. The function loads these files and generates a data structure suitable for filtering and plotting. It returns a object of class cell.data that contains all the required information for the analysis. All the functions included in the package operate over this object, and its components should not be modified directly, but through the provided functions.

Quality Control and Filtering Cells

The algorithm used by Cell-ID to find the cells can occasionally make mistakes in the assignation of the cell boundaries and produce badly found and spurious cells (i.e. image structures erroneously scored as cells). Further more, the program does not discriminate out of focus and dead cells.

Normally you will want to get rid of all the spurious, badly found, out of focus and dead cells (referred collectively as 'bad' cells), which would constitutes a 'quality control' of the data. The R package contains some functions to aid in this process.

`QC.filter`: applies quality control filters over the data. The purpose of this function is to eliminate from the dataset 'bad' cells. You should not use this function to differentiate sub-groups of 'good' cells. This function treats a cell in different time points independently (i.e. it operates on registers of the dataset). To eliminate cells that are not scored in all the time frames, call `update.n.tot` and then filter by n.tot. Filters can be undone by `QC.undo`, or reseted by `QC.reset`. Use `summary.cell.data` to see a summary of the applied filters.

Plotting the Data

For plotting the data you can use the package plotting functions `cplot` and `cplotmeans`, which are simple wrappers over the **ggplot2** package functions.

Data Manipulation

Some common manipulations you can apply over a cell.data object are subsetting (`subset.cell.data`) which returns a cell.data object, extraction (`[[.cell.data`) and aggregation (`aggregate.cell.data`) which return a data.frame.

Author(s)

Alan Bush Maintainer: Alan Bush <abush@fbmc.fcen.uba.ar>

References

`http://www.df.uba.ar/lbms`

See Also

EBImage ggplot2

ACL394 *Time Dependent Dose Response of Yeast Cells to Mating Pheromone*

Description

This datasets was generated by Cell-ID, from an experiment done in 2004 by Alejandro Colman-Lerner and Andrew Gordon at the Molecular Science Institute (MSI). *Saccharomices cerevisiae* yeast cells of strain TCY3154 (MATa, bar1, prm1::Pprm1-YFP::HIS+, trp1::Pact1-CFP::TRP1) where stimulated with different doses of alpha-factor pheromone 10 minutes before the first time point. Images where adquired every 15 minutes for 3.5 hours. In the dataset there are 3 positions per treatment. The strain TCY3154 was derived from ACL394, a W303 derivative.

data(ACL394) loads the unfiltered dataset, while data(ACL394filtered) loads the dataset with filters applied.

pos1.cell.counter is a data.frame as returned by ImageJ's plugin Cell Counter. See cell-counter for more detail.

pdata is a data.frame with the description of each position.

Usage

```
X
```

Format

a cell.data object

References

Colman-Lerner et al. (2005). "Regulated cell-to-cell variation in a cell-fate decision system." Nature 437(7059):699-706.

aggregate *Compute Summary Statistics of Cell Data Subsets*

Description

Splits the data into subsets, computes summary statistics for each, and returns the result in a data frame

Usage

```
## S3 method for class 'cell.data'
aggregate(x, form.by, ..., FUN=mean, subset=TRUE, select=NULL
  ,exclude=NULL, QC.filter=TRUE)
```

Arguments

x	cell.data object
form.by	either a formula or variables to split data frame by, as quoted variables or character vector
...	further arguments passed to or used by methods
FUN	a function to compute the summary statistics which can be applied to all data subsets
subset	a boolean vector of length equal to the number of rows of the dataset, or a conditional statement using the datasets variable, that specifies which registers should be included
select	character vector defining variables names to be included in the returned data.frame
exclude	character vector defining variables names to be excluded from the returned data.frame
QC.filter	a boolean value indicating if the quality control filter should be applied over the data

Details

aggregate is a generic function. This version applies to cell.data objects. Two notations are allowed. If the second argument form.by is a formula it should be of the form cbind(measure.var1,measure.va Note that this notations differs from the one used by reshape.cell.data If the second argument form.by are quoted variables or a character vector with variable names, these variables are taken as group.vars to split the dataset. The measure variables over which to apply FUN should be selected using the select and exclude arguments.

Value

a data frame with columns corresponding to the grouping variables followed by aggregated columns of the measure variables.

Author(s)

Alan Bush

See Also

aggregate

Examples

```
#load example dataset
data(ACL394)

#aggregate by pos and calculate mean f.tot.y
aggregate(X,.(pos),select="f.tot.y")

#do the same aggregation using the formula notation
aggregate(X,f.tot.y~pos)

#aggregate by pos and t.frame
aggregate(X,.(pos,t.frame),select="f.tot.y")
aggregate(X,f.tot.y~pos+t.frame) #formula notation
```

```
#aggregate several variables
aggregate(X,.(pos),select="f.tot.?") # using wildcard pattern matching
aggregate(X,cbind(f.tot.y,f.tot.c)~pos) #formula notation

#subset before aggregating
aggregate(X,.(pos),select="f.tot.y",subset=t.frame==13)

#calculate the median instead of the mean
aggregate(X,.(pos),select="f.tot.y",FUN=median)

#dont apply the QC filter to the daset before aggregation
aggregate(X,.(pos),select="f.tot.y",QC.filter=FALSE)
```

`append` *Append Variables*

Description

This functions append some calculated variables to the cell.data object

Usage

```
append.z.scan(X
  ,fun.z.scan=function(x)(as.numeric(as.factor((x-x%%100)/100)))
  ,fun.z.slice=function(x)(x%%100)
  ,fun.oif=function(x)((x-x%%10000)/10000)
  ,TIME.TOKEN="time",TIME.DIGITS=5
  ,channel=X$channels$name[1])

append.in.focus(X,focus.var,in.focus.var="in.focus")

append.anular.y(X)
append.anular.r(X)
append.anular.c(X)

append.memRec.y(X)
```

Arguments

`X`	cell.data object
`focus.var`	character name of variable used to focus
`in.focus.var`	character name of appended variable
`fun.z.scan`	function used to extract the z.scan from the image time token
`fun.z.slice`	function used to extract the z.slice from the image time token
`fun.oif`	function used to extract the oif number from the image time token
`TIME.TOKEN`	Image time token
`channel`	character specifying the channel to use to extract the relevant information from the filenames
`TIME.DIGITS`	numeric digits of the time token

Details

append.z.scan appends the variables 'z.scan', 'z.slice' and 'oif' to the dataset. 'z.scan' indicates the z stack a time frame blongs to. 'z.slice' indicates the slice within a z.scan. 'oif' indicates from wich file the image comes from. append.in.focus appends a boolean vector that is TRUE when the position mean of the selected focus.var is maximum within a z.scan append.anular functions append the variables 'f.p1', 'f.m0', 'f.m1', 'f.m2', 'f.m3' and the respective areas to the dataset, in a channel specific manner. append.memRec.y calcualates the membrane recruitment observable 'f.obs.y', for YFP channel

Value

returns a cell.data object, with appended variables

Author(s)

Alan Bush

See Also

transform.cell.data

Examples

```
## Not run:
X<-append.anular.y(X)
X<-append.memRec.y(X)
X<-append.z.scan(X)
X<-append.in.focus(X,"f.obs.y")

## End(Not run)
```

---

append.oif *Append Variables from OIF files*

Description

This functions create new variables containing information of OIF (Olympus Image Format) files.

Usage

```
append.oif.time(X,OIF.date='OIF-date.txt',path=getwd(),pos.digits=2
  ,oif.digits=2)
```

Arguments

X	cell.data object
OIF.date	string containing the name of the OIF-date file (see details).
path	path to the OIF-date file (see details). Working directory is used by default.
pos.digits	Integer indicating the number of digits expected for position.
oif.digits	Integer indicating the number of digits expected to specify the file number within a position.

Details

This function can be used to add the time information of a OIF (Olympus Image Format) file to the cell.data object. To do so you first have to generate a single text file with the time information of all the .oif files. To create this file (OIF-date.txt) in Windows you can use the following scripts

oif2txt.bat: `for %%i in (*.oif) do type %%i > %%~ni.txt`

selectLineFromOif.bat: `sfk filter -ls+"ImageCaputre" -file .txt > OIF-date.txt`

The first one changes the encoding of the .oif files, from Unicode to ASCII. The second one uses sfk (http://swissfileknife.sourceforge.net/) to extract the time information from each oif file. The OIF-date.txt file should look like this:

01_01_YPP3662_XYZ.txt :
ImageCaputreDate='2011-08-20 11:15:59'
ImageCaputreDate+MilliSec=984

The oif filenames are expected to be of the form ??_??_* where ? are digits [0-9]. The digits before the underscore specify the position, and the digits after the underscore specify the "oif number" (number of file within a position). `pos.digits` and `oif.digits` specify the expected digits for these numbers respectively.

Value

returns a cell.data object, with appended variables

Author(s)

Alan Bush

See Also

`merge.cell.data`

Examples

```
## Not run:
X<-append.oif.time(X)

## End(Not run)
```

---

as.cell.data *Coerce to Cell Data*

Description

Coerces a list or data.frame to a cell.data object

Usage

```
as.cell.data(X,...)

## S3 method for class 'list'
as.cell.data(X,path.images=NULL,...)

is.cell.data(X)
```

Arguments

X	list to be coerced to (or test for) cell.data object
path.images	string containing path to the image files
...	additional arguments to be passed to or from methods

Details

as.cell.data coerces objects to class cell.data. If a list is coerced, it is expected to have components 'data', 'bf.fl.mapping' and others. It is specially usefull to coerce data loaded with previous versions of Rcell. is.cell.data test if a object inherits from class cell.data

path is used to update the path of the image files.

Value

a cell.data object

Author(s)

Alan Bush

See Also

load.cellID.data

Examples

```
#load example dataset
data(ACL394)

#transforming dataset to list
Xlist<-as.list(X);class(Xlist)<-"list";

#re-coerce to cell.data
Y<-as.cell.data(Xlist)
```

`as.data.frame` *Coerce to a Data Frame*

Description

Function for extracting a (subset) data.frame from a cell.data object

Usage

```
## S3 method for class 'cell.data'
as.data.frame(x,row.names=NULL,optional=FALSE,...,subset=TRUE
  ,select=NULL,exclude=NULL,QC.filter=TRUE)

 ## S3 method for class 'cell.data'
x[[subset=TRUE,select=NULL,exclude=NULL,QC.filter=TRUE,...]]

 cdata(x,subset=TRUE,select=NULL,exclude=NULL,QC.filter=TRUE,...)
```

Arguments

`x`	cell.data object
`subset`	a boolean vector of length equal to the number of rows of the dataset, or a conditional statement using the dataset¥s variable, which specifies which registers should be included in the returned data.frame
`select`	character vector defining variables names to be included in the returned data.frame
`exclude`	character vector defining variables names to be excluded from the returned data.frame
`QC.filter`	a boolean value indicating if the quality control filter should be applied over the data
`...`	further arguments passed to or used by methods
`row.names`	further arguments passed to or used by methods
`optional`	further arguments passed to or used by methods

Details

as.data.frame.cell.data coerces a cell.data object to a data.frame, subsetting it as defined by the other arguments. This function will be called when the generic function `as.data.frame` is applied over a cell.data object.

The extract ('[[') operator is an alias to this function.

`select` and `exclude` can be used to choose which variables should be included in the returned data.frame. Wildcard patterns (e.g. 'f.*.y') and keywords (e.g. 'all', 'id.vars', 'YFP', etc.) can be used as components of these arguments. Use `summary.cell.data` to see available variables and keywords. Variable names starting with '-' in `select` will be excluded from the data.frame.

Value

A data.frame, subset as specified by the functions arguments.

Author(s)

Alan Bush

See Also

as.data.frame

Examples

```
#load example dataset
data(ACL394)

#extract the dataset to a data.frame
df<-as.data.frame(X)
df<-X[[]]

#extract a subset of the data.frame
df<-X[[t.frame==13,]]

#extract a selected group of variables
df<-X[[,c("id.vars","f.tot.?","a.tot")]]
#note the use of keywords, patterns and variable names

#extract the dataset without applying the QC filter
df<-cdata(X,QC.filter=FALSE)
```

---

cell-counter *Map Cell Counter Tags to Cells*

Description

This functions maps the tags generated by ImageJ plugin Cell Counter to the cells in a cell.data object

Usage

```
map.cells.points(X,cell.counter,pos=NULL,t.frame=0,...
  ,radius=10,var.name="tag",init.value=NA,map.to.all.t=TRUE)

cardinality.plot(X,cell.counter,pos=NULL,t.frame=0,...
  ,max.radius=30,max.cardinality=3)
```

Arguments

X	cell.data object
cell.counter	data.frame loaded from Cell Counter output, or a list containing such data.frames. The list index should corresponds to the position number of the image
pos	if cell.counter is a data.frame, the position it corresponds to. If cell.counter is a list this argument is not used.
t.frame	the time frame of the dataset to use for the mapping

`...`	arguments to be passed to private methods. subset and QC.filter can be specified
`radius`	integer radius used for the mapping. It should be a value in the plateau of `cardinality.plot`
`var.name`	name of the new variable with the tags
`init.value`	value assign to cells that where not mapped to any point
`map.to.all.t`	boolean. if TURE the new variable will be assign to all t.frames, if FALSE only to `t.frame` time frame
`max.radius`	maximum radius to calculate the cardinality
`max.cardinality`	maximum cardinality to show in the plot

Details

This functions are used to merge additional data to the Cell-ID dataset. Tags are assign manually to cells using 'Cell Counter' plugin of ImageJ. To do so open the BF or fluorescent image in ImageJ, select Plugins > Analyse > Cell Counter. Click on 'initialize'. A copy of your image should appear. Select 'Point Selection' from ImageJ buttons, and click on the counter type in the Cell Counter window. Mark the cells with the correspondent tag (1,2,3..). Make sure to put the tag close to the center of the cell.

When finished click 'Measure...' in the CellCounter windows. A table with the point Type, X and Y position should appear. Save it as a .txt file and take note of the path (for example 'c:/data/TFP_Pos1_time1.CellCounter.txt'). If you want to save the image with the tags, click on 'Export Image' in CellCounter, and save the image.

Back in R, load the CellCounter table

```
tags1<-read.table('c:/data/TFP_Pos1_time1.CellCounter.txt',head=T)
```

The mapping between the points and cells will be done based on the XY position. We need a cut-off radius. If the distance between cell center (determined by Cell-ID) and the point (from CellCounter) is less than the cut-off, the point type will be assigned to the cell. To choose the correct cut-off radius use the `cardinality.plot` function

```
cardinality.plot(X,tags1,pos=1,t.frame=0)
```

Choose a radius in the plateau of the cardinality=1 curve (one to one mapping between cells and points). If cut-off radius is to high, some ambiguities will appear in the cell assignation. If cut-off radius is two low, some points won't be assigned to their correspondent cells. Usually radius=10 is a good value. Finally do the mapping

```
X<-map.cells.points(X,tags1,pos=1,t.frame=0,radius=10,var.name='tag.type')
```

This will add a new variable to the cell.data object named 'tag.type', with the correspondent tag number for each cell of position 1. The same tag will be added to all time points. t.frame specifies which t.frame is used for CellCounter tag assignation.

If you want to add tags for more than one position, you have to two options. The first one is to follow the steps shown above for other positions. Note that usually the same radius works for all positions.

```
tags2<-read.table('c:/data/TFP_Pos2_time1.CellCounter.txt',head=T)
X<-map.cells.points(X,tags2,pos=2,t.frame=0,radius=10,var.name='tag.type')
```

If you want to reset the assign tags, use the map.cells.points with the 'add' argument set to FALSE. A second option is to construct a list with each 'Cell Counter' data.frame as an element. The name

of the element in the list should correspond to its position. For example if you have the data.frames of positions 1, 2 and 5 loaded in the variables tags1, tags2 and tags5 create a new list and use it as the cell.counter argument.

```
tags.list<-list(tags1,tags2,tags5)
names(tags.list)<-c("1","2","5")
X<-map.cells.points(X,tags.list,t.frame=0,radius=10,var.name='tag.type')
```

Value

a cell.data object with the tags from Cell Counter merged to the data.frame

Author(s)

Alan Bush

See Also

transform.cell.data, merge.cell.data

Examples

```
#load the example dataset
data(ACL394)

#pos1.cell.counter is a cell counter output file for position 1
str(pos1.cell.counter)

#plotting cardinality
cardinality.plot(X,pos1.cell.counter,pos=1)

#do the mapping
X<-map.cells.points(X,pos1.cell.counter,pos=1,radius=10,var.name="cell.type")

#use the new variable for plotting
cplot(X,f.tot.y~t.frame,color=cell.type,subset=pos==1)
```

---

cell.hclust *Hierarchical Clustering of Cell Data*

Description

Hierarchical cluster analysis on cells of a cell.data object

Usage

```
cell.hclust(X,select,metric="cosangle",method="average",plot="heatmap"
  ,main=NULL, heatmap.col=colorRampPalette(c("green", "black", "red")
    ,space="rgb",bias=2)(128)
  ,cutree="none",cutree.args=list(h=0.5),min.cluster.size=20
  ,formula=ucid ~ variable + t.frame
  ,subset=TRUE,exclude=NULL,QC.filter=TRUE
  ,col.select=NULL,col.exclude=NULL,labRow=NA,...)
```

Arguments

X	cell.data object
select	character vector defining variables names (before reshaping) to be included for the clustering
metric	character string specifying the metric to be used for calculating dissimilarities between vectors. The currently available options are "cosangle" (cosine angle or uncentered correlation distance), "abscosangle" (absolute cosine angle or absolute uncentered correlation distance), "euclid" (Euclidean distance), "abseuclid" (absolute Euclidean distance), "cor" (correlation distance), and "abscor" (absolute correlation distance).
method	the agglomeration method to be used. This should be (an unambiguous abbreviation of) one of "ward", "single", "complete", "average", "mcquitty", "median" or "centroid".
plot	type of plot to be printed to the active device. Currently available options are "heatmap" or "none".
main	title for the plot. If NULL metric, clsuter method and tree cut method are specified
heatmap.col	vector specifying colors to be used as the heatmap palette
cutree	method use to cut the hierarchical clustering tree. Currently available options are "none" or "height"
cutree.args	list of arguments to be passed to the cutree method
min.cluster.size	minimal amount of cells of a cluster
formula	casting formula, see details for specifics
subset	a boolean vector of length equal to the number of rows of the dataset, or a conditional statement using the dataset¥s variable, that specifies which registers should be included
exclude	character vector defining variables names to be excluded from the clustering
QC.filter	a boolean value indicating if the quality control filter should be applied over the data
col.select	character vector defining variables names (after reshaping) to be included for the clustering. Wildcard patterns are also accepted
col.exclude	character vector defining variables names (after reshaping) to be excluded of the clustering. Wildcard patterns are also accepted
labRow	character vectors with row labels to use; if NA (the default) no row labels are shown
...	further arguments for heatmap or plotting function

Details

This functions does a hierarchical clustering of the cells. For that it first reshapes the data with a call to creshape. The formula argument should a have a single variable in the left term (usually 'ucid' or 'cellID').

The function then calculates a distance matrix using the function distancematrix of the hopach package. The function hclust is used to calculate the clustering. If a cutree method is specified, the cells are grouped into clusters. The function then plots a heatmap to the current device.

Value

a (invisible) list containing elements $data, $matrix, $dist, $hclust and $cell.subtree. $data is the reshaped data.frame. $matrix contains the same information as $data, coerced to matrix. $dist contains the distance matrix calculated with the method specified in `metric`. $hclust contains the output of the call to `hclust`. $cell.subtree contains a data.frame with the subtree that each cell belongs to.

Author(s)

Alan Bush

See Also

`distancematrix`,`hclust`,`heatmap`

Examples

```
#load example dataset
#warning: Any object named 'X' will be replaced
data(ACL394filtered)

#Heriarchical clustering of cells by f.tot.y time course,
#using cosangle (uncentered correlation) metric and average linkage method.
cell.hclust(X,"f.tot.y")

#Heriarchical clustering of cells by f.tot.y time course,
#using euclid metric and complete linkage method.
cell.hclust(X,"f.tot.y",metric="euclid",method="complete")

#Cut the tree at constant height and show the clusters
cell.hclust(X,"f.tot.y",cutree="height",cutree.args=list(h=0.005))

#redefining the formula, plot against time in minutes
X<-transform(X,time.min=10+t.frame*15) #calculating the time of each t.frame
cell.hclust(X,"f.tot.y",formula=ucid~variable+time.min)
```

`cell.image` *Get Cells Images*

Description

Retrieves the images from single cells in an cell.image object

Usage

```
get.cell.image(X,...)

## S3 method for class 'cell.data'
get.cell.image(X,subset=NULL,channel.subset=NULL,channel=NULL
  ,group=NULL,N=7,select=NULL,exclude=NULL,QC.filter=TRUE,box.size=20,...)
```

```
## S3 method for class 'data.frame'
get.cell.image(X,box.size=20,...)

## Default S3 method:
get.cell.image(X,box.size=20,...)

## S3 method for class 'cell.image'
summary(object,...)

## S3 method for class 'summary.cell.image'
print(x,...)

## S3 method for class 'cell.image'
print(x,nx=ceiling(sqrt(length(x))),...)

img.desc(X)

is.cell.image(X)
```

Arguments

X	cell.data object or data.frame that specifies the images
subset	logical expression indicating elements or rows to keep. Don't specify channel here.
channel.subset	logical expression to specify which image to retrieve with channel and t.frame variables.
channel	character vector of channels to retrieve.
group	character vector or quoted names of variables who's interaction define the groups from which select N random cells.
N	Number of random cells to select from each group. If NULL all cells are selected
select	character vector defining variables names to be included in the returned cell.image object
exclude	character vector defining variables names to be excluded from the returned cell.image object
QC.filter	a boolean value indicating if the quality control filter should be applied over the data
box.size	size in pixels of the image containing the cells. This specifies the 'radius', i.e. the image will be a square of length 2*box.size+1
...	further arguments for methods
object	cell.image object to summarize
x	object to print
nx	number of columns in the image tile

Details

get.cell.image is a generic method that returns a cell.image object.

If `get.cell.image` first argument is a data.frame, it should contain the columns path, image, xpos and ypos.

If the first argument when calling `get.cell.image` is a cell.data object, further arguments specify which images will be selected. The `subset` arguments filters the dataset as in other functions. If some variables are specified in `group`, the data is split in groups defined by these variables, and from each group `N` cells are selected at random. The `channel` argument specifies which channels to show. If a more complex image selection is required, you can use the `channel.subset` argument. For example if you want to see the BF only for the first t.frame, and then only the YFP channel, you can use `channel.subset=channel=='YFP'|(t.frame==0&channel=='BF')`

`img.desc` returns a data.frame describing each image of the cell.image object

Value

a cell.image object. This object is basically a list who's elements are the cropped images of single cells. It has a attribute named 'img.desc' that is a data.frame with the image index (img.index) and description of all the components of the objects.

Author(s)

Alan Bush

See Also

EBImage

Examples

```
#load example dataset
data(ACL394filtered)

#select N=3 cells images from each pos (group),
#from the first t.frame and pos 1,8,15,22,29.
ci<-get.cell.image(X,subset=match(pos,c(1,8,15,22,29),nomatch=0)>0&t.frame==11,
group=.(pos),N=3,channel=c('BF.out','YFP'))
if(interactive()) ci #print the cells images
summary(ci) #get a summary of the content
img.desc(ci) #get the image description data.frame

#select the first 4 t.frames for YFP, and the first t.frame for BF
ci<-get.cell.image(X,subset=pos==29,group='pos',
channel.subset=channel=='YFP'|(t.frame==11&channel=='BF'))
if(interactive()) print(ci)
```

---

cimage *Plot cell images*

Description

Arranges cells images in a plot

Usage

```
cimage(X,...)

## S3 method for class 'cell.data'
cimage(X,formula=NULL,facets=NULL,time.var=c('time','t.frame')
  ,select=NULL,exclude=NULL,normalize.group='channel',...)

## S3 method for class 'cell.image'
cimage(X,formula=NULL,facets=NULL,scales='fixed'
  ,nx=NULL,ny=NULL,facets.nx=NULL,facets.ny=NULL
  ,bg.col='gray',border=1,facets.border=1
  ,font.size=14,font.col='black',display=interactive(),...)

## Default S3 method:
cimage(X,...)
```

Arguments

X	cell.data or cell.image object to plot
formula	formula of the form 'var1+var2~var3' specifying how the images are to be ordered. See details.
facets	formula of the form 'var1+var2~var3' specifying how to facet the plot. See details.
time.var	variables that indicate time and should be excluded from the grouping variables. See get.cell.image
select	character vector defining further variables that are required for the plot
exclude	character vector defining variable names to be excluded
normalize.group	variable names that define groups of images that should be normalized together
scales	either 'fixed' or 'free' axis for each facet
nx	number of columns of images within each facet. Used with formula '~var1' or 'var1~.'
ny	number of rows of images within each facet. Used with formulas '~var1' or 'var1~.'
facets.nx	number of columns of facets. Used with facets '~var1' or 'var1~.'
facets.ny	number of rows of facets. Used with facets '~var1' or 'var1~.'
bg.col	The background color of the plot
border	the width in pixels of the border between images
facets.border	the width in pixels of the border between facets
font.size	The size of the font to use, in pixels
font.col	The color of the font to use
display	boolean indicating if the created image should be displayed
...	further arguments for methods. cimage calls get.cell.image, so all the arguments of this function are available.

Details

Read the cimage vignette for a tutorial on how to use this function: vignette('cimage')

`cimage` is a generic method that returns a 'Image' object, from EBImage package.

If `cimage`'s first argument is a cell.data object, it first calls `get.cell.image` and then the `cimage` method for cell.image objects. This function arranges the images of single cell according to the `formula` and `facets` arguments, and adds appropriated axis to the image.

For example, formula=channel~t.frame, will arrange different channels as rows and t.frame as columns. You can use several variables per term, for example formula=channel~pos+t.frame will arrange the columns first by position, and within each position by t.frame. The variable to the right varies faster than the one to the left. If only the right term of the formula is defined, as in formula=~t.frame, the images are 'wrapped' around, attempting to create a square plot. `nx` and `ny` can be used to define the number of columns or rows respectively. The special argument ... can be used to indicate the samples within a group, for example formula=...~t.frame. The `facets` argument works in a similar way.

Value

The function returns an invisible 'Image' object of the EBImage package. Use display to render the image or writeImage to save it.

Author(s)

Alan Bush

See Also

EBImage,display

Examples

```
#load example dataset
data(ACL394filtered)

#display timecourse strip of cell 5 of pos 29, channels BF and YFP
if(interactive()) cimage(X,channel~t.frame,subset=pos==29&cellID==5,channel=c('BF','YFP')

#display 7 cells (default value for N) of pos 29
if(interactive()) cimage(X,...+channel~t.frame,subset=pos==29,channel=c('BF','YFP'))

#display 3 cells from each pos in a different facet
if(interactive()) cimage(X,channel~...,facets=~pos,channel=c('BF.out','YFP'),N=3,
    subset=t.frame==11&match(pos,c(1,8,15,22,29),nomatch=0)>0)

#select one BF and many YFP images
if(interactive()) cimage(X,...~channel+t.frame,subset=pos==29,N=3,
channel.subset=channel=='YFP'|(channel=='BF.out'&t.frame==11))
```

`cplot` *Plotting Cell Data Objects*

Description

Plotting functions for cell.data objects. These functions are wrappers over the functions of ggplot2 package.

Usage

```
cplot(X=NULL, x=NULL, subset = NULL, y=NULL, z=NULL, ...
  ,facets = NULL, margins=FALSE, geom = "auto"
  ,stat=list(NULL), position=list(NULL), log = "", as.factor="as.factor"
  ,xlim = c(NA, NA), ylim = c(NA, NA), xzoom = c(NA,NA), yzoom = c(NA,NA)
  ,xlab = deparse(substitute(x)), ylab = deparse(substitute(y)), asp = NA
  ,select = NULL, exclude = NULL, QC.filter = TRUE, droplevels=TRUE
  ,main = NULL, add = FALSE, layer = FALSE)

clayer(...,geom="auto")

cplotmeans(...,geom=c("point","errorbar","line"))

clayermeans(...,geom=c("point","errorbar","line"))

## S3 method for class 'cell.data'
plot(x,y,...)
```

Arguments

`X`	cell.data object
`x`	either a variable symbol or expression, or a formula of the form y~x or ~x
`subset`	a boolean vector of length equal to the number of rows of the dataset, or a conditional statement using the dataset¥s variable, which specifies which registers should be included in the plot
`y`	a variable symbol or expression to be plot in the y axis. Ignored if x is a formula. A vector of symbols is allowed
`z`	a variable symbol specifying the "z" aesthetic mapping
`...`	other arguments passed on to the geom functions
`facets`	faceting formula to use
`margins`	whether or not margins will be displayed
`geom`	geom to use (can be a vector of multiple names)
`stat`	statistic to use (can be a vector of multiple names)
`position`	position adjustment to use (can be a vector of multiple names)
`log`	which variables to log transform ("x", "y", or "xy")
`as.factor`	variable names (wildcard pattern or keyword) to be treated as factors

xlim	limits for x axis c(min,max) (filters the x variable BEFORE applying the stat transformation)
ylim	limits for y axis c(min,max) (filters the y variable BEFORE applying the stat transformation)
xzoom	zoom range for x axis c(min,max) (resizes the plotting region AFTER the stat transformation)
yzoom	zoom range for y axis c(min,max) (resizes the plotting region AFTER the stat transformation)
xlab	character vector or expression for x axis label
ylab	character vector or expression for y axis label
asp	the y/x aspect ratio
select	character vector defining variables names to be included in the returned ggplot object, beside the ones required for the plot
exclude	character vector defining variables names to be excluded from the returned ggplot object
QC.filter	a boolean value indicating if the quality control filter should be applied over the data before plotting
droplevels	boolean specifying if the empty levels of factors should be dropped (removed)
main	character vector or expression for plot title
add	the plot is added as a layer to the last plot (returned by last_plot)
layer	boolean. If TRUE a layer is returned instead of a new ggplot object. Mutually exclusive with add

Details

Read the cplot vignette for a tutorial on how to use this function: vignette('cplot')

cplot is a wrapper over the functions of ggplot2 package from Hadley Wickham. It is based on qplot and keeps many of its arguments. The main differences between cplot and qplot are the following:

- cplot's first argument is a cell.data object (or a data.frame)
- the 'x' and 'y' aesthetic mapping can be specified by a formula in cplot
- a vector of variables can be specified for 'y' aesthetic mapping. This produces a data restructuring and sets the color aesthetic to variable
- variables selected by as.factors are coerced to factors before plotting
- the plotting region can be easily specified with xzoom and yzoom. Useful when stat='summary'.
- a subset of the dataset can be performed before plotting
- only the required variables for the plot are included in the ggplot object, thus reducing the memory space it requires. Additional variables can be included with the select and exclude arguments.
- if a logical QC variable is present in the dataset, it is used to filter it before plotting
- unused levels of factors can be drop with droplevels
- the specified plot can be returned as a layer to add to other plots with the '+' operator

clayer is a wrapper for cplot with `layer`=TRUE. This function returns a layer that can be added to other ggplot objects with the '+' operator.

cplotmeans (alias cplotmean) is a wrapper over cplot with `stat`='summary' and `fun.data`='mean_cl_normal'. This function plots the mean and confidence limits for the mean of the data, grouped by levels of the x variable. The default confidence interval is of 95%, and can be modified with the `conf.int` argument (passed to `smean.cl.normal`).

clayermeans (alias clayermean) is a wrapper over cplot with `stat`='summary', `fun.data`='mean_cl_normal' and `layer`=TRUE.

plot.cell.data is a wrapper over cplot. It only accepts formula notation for the 'x' and 'y' aesthetics. It can be called by `plot` over a cell.data object.

Value

a ggplot object or a list specifying plots layers

Author(s)

Alan Bush

References

H. Wickham. ggplot2: elegant graphics for data analysis. Springer New York, 2009.

See Also

`qplot`,`ggplot`

Examples

```
#load example dataset
data(ACL394)

#plotting YFP vs CFP fluorescence
cplot(X,f.tot.y~f.tot.c)

#reduce point size (and alpha blending) to eliminating overplotting
cplot(X,f.tot.y~f.tot.c,size=0.5) #add alpha=0.3 for 30% transparency

#subset the data before plotting
cplot(X,f.tot.y~f.tot.c,subset=t.frame==13)

#color by pos variable
cplot(X,f.tot.y~f.tot.c,subset=t.frame==13,color=pos)

#map the size aesthetic to the the cell area a.tot
cplot(X,f.tot.y~f.tot.c,subset=t.frame==13,color=pos,size=a.tot)

#adding description of the positions for futher plotting
# (AF.nM: dose of alpha-factor yeast pheromone in nM)
X<-merge(X,data.frame(pos=1:35,AF.nM=rep(c(1.25,2.5,5,10,20),each=7)))

#plot time course for f.tot.y and facet by pheromone dose
cplot(X,f.tot.y~t.frame,facets=~AF.nM)

#jittering the points to reduce overplotting
```

```
cplot(X,f.tot.y~t.frame,facets=~AF.nM,size=0.5,geom="jitter")

#adding per t.frame mean to prevoius plot
cplot(X,f.tot.y~t.frame,facets=~AF.nM,size=0.5,geom="jitter")+
  clayermean(color="red")

#plot means for each dose in the same plot
cplotmean(X,f.tot.y~t.frame,color=AF.nM,as.factors="AF.nM",yzoom=c(0,6.2e6))

#plotting histograms
cplot(X,~f.tot.y)

#map fill aesthetic to AF.nM variable coerced as factor
cplot(X,~f.tot.y,fill=AF.nM,as.factors="AF.nM")

#use position 'dodge' instead of 'stack'
cplot(X,~f.tot.y,fill=AF.nM,as.factors="AF.nM",position="dodge")
```

---

load.cellID.data *Load Cell-ID Data*

Description

load.cellID.data searches a specified directory (the working directory by default) for folders that match a customizable pattern, usually PositionXX where XX is the position number. This folders should contain the Cell-ID output files output_all and the output_bf_fl_mapping for each position. The function loads this files and generates a data structure suitable for filtering and plotting. The function returns a cell.data object that contains all the required information for the analysis. All the functions included in the package operate over this object, and its components should not be modified directly, but through the provided functions. Remember to assign the returned value to a variable (e.g. X<-load.cellID.data())

Usage

```
load.cellID.data(pattern="^[Pp]{1}os[:alpha:]*[:digit:]*",path=getwd()
  ,basename="out",select=NULL,exclude=NULL,load.vars="all")
```

Arguments

pattern	regular expression (see regexp) pattern of the position folders, where the images and cell ID output files for each position are stored.
path	character containing path from where to apply the pos.pattern to the existing folders. It should point to the folder that contains the PosXX folders.
basename	character containing basename of the cell ID output files, should match the -o option passed to cellID when executed. 'out' by default.
select	character vector defining variables names to be included in the cell.data object
exclude	character vector defining variables names to be excluded of the cell.data object
load.vars	character specifying which variables or group of variables of the Cell-ID out_all file should be loaded.

Details

reads Cell ID output files (basename)_all in folders that match pattern in path and loads them into a cell.data object.

It searches for the output_all files in folders of the form specified by pattern (regular expression). If the folder has a numeric value in its name that number is taken as the position index (for example pos01 is given the index 1) If no numeric value is found in the folder name, then a ordinal index is assign.

Possible values for load.vars are 'all', 'fl' or 'fluorescence', 'bg' or 'background', 'calc', 'morph' or 'morphological', 'vac' or 'vacuole', 'nucl' or 'nuclear', 'disc'. The group of variables can be specified in either a positive form (i.e. '+fl+bg+morph') or in a negative form (i.e. '-nucl-vac'). Combination of positive and negative form is not allowed. A character vector containing the variables names of the out_all file is also allowed. The selection of variables is done before restructuring, so the variable names should correspond to those of the out_all files. Using this argument can be useful if memory issues arise.

Alternatively `select` and `exclude` can be used to subset the dataset. This arguments are applied after the reshaping, so variables names as returned by `summary.cell.data` are used. Wildcard patterns (e.g. 'f.*.y') and keywords (e.g. 'all', 'id.vars', 'YFP', etc.) can be used as components of these arguments.

Value

a cell.data object

Note

The restructuring of the data involves arranging the information for each time point of each cell into a single row. In the output of Cell-ID this information appears in several rows, one for each channel. The restructured data 'collapses' this rows into a single one, adding and modifying the column names by appending a channel specific postfix. When Cell-ID is run, the images it uses have to be named in a specific way. The first three letters of the image name are used as a channel token, i.e. it identifies the channel. If you have YFP and CFP channels, the images should be named YFP_Position1, YFP_Position2,...,CFP_Position1, CFP_Postion2,... The channel postfix is the shortest unambiguous substring of the channel token in lower case. For example for the tokens 'YFP' and 'CFP', the selected postfix will be 'y' and 'c' respectively.

Author(s)

Alan Bush

See Also

`read.table`,`dir`, `QC.filter`, `summary.cell.data`

Examples

```
## Not run:
setwd(".")  #set the working directory to the folder with your images
X<-load.cellID.data()  #load the dataset to R


## End(Not run)
```

merge *Merge a Data Frame to a Cell Data Object*

Description

Merges the variables in a data.frame to a cell.data object, using common variables to do the merging

Usage

```
## S3 method for class 'cell.data'
merge(x,y,by=NULL,na.rm=FALSE,...)

load.pdata(X,pdata="pdata.txt",by=NULL,path=getwd())
```

Arguments

X	cell.data object
x	cell.data object
y	a data.frame with at least one common variable with x
by	character vector indicating which variables to use for the merging
na.rm	should NAs be removed before merging
pdata	either a string with the filename of a tab delimited text file containing the data to be merged, or a data.frame to merge
path	string containing the path to the location of the tab delimited file to be loaded
...	additional arguments to be passed to or from methods

Details

merge is used to add the variables in a data.frame to the cell.data object. It uses common variables to do the merging. The variables can be specified with the by argument.

load.pdata is a wrapper over merge, used to load position information to the cell.data object. By default it looks for a file named 'pdata.txt' in the working directory. This file should have a 'pos' column.

Value

a cell.data object with the merged variables.

Author(s)

Alan Bush

See Also

merge

Examples

```
#load example dataset
data(ACL394)
#creating data frame with information about each poistion
#AF.nM: dose of alpha-factor yeast pheromone in nM
pdata=data.frame(pos=1:35,AF.nM=rep(c(1.25,2.5,5,10,20),each=7))

#merging the data frame with the cell.data object
X<-merge(X,pdata)
```

`misc` *Miscellaneous Functions*

Description

Miscellaneous functions to do stuff in less lines

Usage

```
paste_data_error(data,error,error.signif=1)
paste_parameter(fit,param,error.signif=1)
paste_intercept_slope(fit,error.signif=1)
paste_EC50_n(fit,leading.str="",error.signif=2)
vplayout(x, y)
```

Arguments

`data`	a numeric vector of values
`error`	a numeric vector of errors for `data` values
`error.signif`	number of significant digits for the error
`fit`	an object of class 'lm' or 'nlm'
`param`	character name of the parameter from `fit` to paste
`leading.str`	string to paste before the data and error
`x`	x index of grid to use to print the ggplot2 figure
`y`	y index of grid to use to print the ggplot2 figure

Details

the `paste_` functions are used to paste a value and its error (or uncetainty) in resonable way.

Value

a character vector with the data and error

Author(s)

Alan Bush

See Also

transform.cell.data

Examples

```
paste_data_error(1.0,0.01)

#put several figures in a page
data(ACL394)
grid.newpage() #create a new plot
pushViewport(viewport(layout = grid.layout(1, 2))) #define the grid for the plots
print(cplot(X,f.tot.y~pos), vp = vplayout(1, 1))
print(cplot(X,f.tot.y~a.tot,color=pos), vp = vplayout(1, 2))
```

QC.filter *Quality Control Filter*

Description

Create, undo, reset and execute quality control filters

Usage

```
QC.filter(X, filter, subset=NULL)

QC.undo(X)

QC.reset(X)

QC.execute(X)
```

Arguments

X	the cell.data object as returned by load.cellID.data make sure to save the object when it's returned by the function i.e. do the calls as X=QC.filter(X,...))
filter	a boolean vector of length equal to the number of rows of the dataset, or a conditional statement using the datasetYens variable, that specifies which rows pass the quality control (TRUE), and which ones don't (FALSE).
subset	a boolean vector of length equal to the number of rows of the dataset, or a conditional statement using the dataset variable, which specifies over which registers filter should be applied.

Details

QC.filter function filters the cells based on a user define boolean vector filter Such vector can be obtained applying logical operations over the vectors (fft.stat<0.2, etc). The purpose of this filter is to eliminate from your dataset spurious, badly found, out of focus and dead cells. This filter is cumulative, meaning that each time one applies a QC.filter function it adds to the previous QC.filter, it does not replaced them. Many functions from the package have a QC.filter argument, that specifies if the created QC filter should be applied to the dataset before the function is executed.

The filter treats the same cells in different time points independently. Don't use this function to select subgroups of cells (see `select.cells`) `summary.cell.data` returns a description of the applied filters.

QC.undo removes the last filter applied. QC.reset eliminates all filters, restoring the cell.data object to its original state. QC.execute permanently eliminates the filtered registers. This is recommended only if you have memory issues.

Value

Returns the cell.data with the specified filter applied.

Note

Some times it is useful to create additional filters to discriminate between cells. Dont use QC.filter for this. You can create a filter with `transform.cell.data` and use the `subset` argument of the function you want to apply.

Author(s)

Alan Bush

See Also

`summary.cell.data`,`transform.cell.data`,`load.cellID.data`

Examples

```
#load example dataset
data(ACL394filtered)

#resetting all the filters
X<-QC.reset(X)

#filtering by fft.stat
cplot(X,~fft.stat) #see what cut to use
X<-QC.filter(X,fft.stat < 0.5) #apply the cut

#filtering by the total number of frames in which a cell appears
cplot(X,cellID~t.frame,fill=f.tot.y,geom="tile",facets=~pos)
X<-update.n.tot(X) #updating n.tot variable
cplot(X,~n.tot) #define where to apply the cut
X<-QC.filter(X,n.tot==14) #keep cells that appear in all t.frames

#exclude cells by ucid (Unique Cell ID)
cplot(X,f.total.y~time.min,facets=~AF.nM,size=0.3,geom="jitter")
#selecting cells that don't respond
c1=select.cells(X,f.total.y<10e4&t.frame>3,n.tot.subset=n.tot>=8)
X<-QC.filter(X,!ucid %in% c1)

#undoing the last filter
X<-QC.undo(X)
```

`reshape.cell.data` *Reshape a Cell Data Object*

Description

Reshapes the data in a cell.data object and returns a data.frame

Usage

```
reshape(data,...)

## S3 method for class 'cell.data'
reshape(data,formula = pos + cellID ~ variable + t.frame
  ,fun.aggregate=NULL, ..., margins=FALSE, fill=NULL
  ,id.vars=NULL, measure.vars=NULL, variable_name = "variable", na.rm = FALSE
  ,subset=TRUE ,select=NULL ,exclude=NULL ,QC.filter=TRUE)
```

Arguments

`data`	cell.data object
`formula`	casting formula, see details for specifics
`fun.aggregate`	aggregation function
`...`	further arguments are passed to aggregating function
`margins`	vector of variable names (can include 'grand_col' and 'grand_row') to compute margins for, or TRUE to computer all margins
`fill`	value with which to fill in structural missing, defaults to value from applying `fun.aggregate` to 0 length vector
`id.vars`	character vector of id variables names, wildcard pattern or keyword. If NULL, will use all variables of the formula.
`measure.vars`	character vector of measure variables names, wildcard pattern or keyword. If NULL, will use all non id.vars variables.
`variable_name`	Name of the variable that will store the names of the original variables
`na.rm`	Should NA values be removed from the data set?
`subset`	a boolean vector of length equal to the number of rows of the dataset, or a conditional statement using the dataset¥s variable, that specifies which registers should be included
`select`	character vector defining variables names to be included in the returned data.frame
`exclude`	character vector defining variables names to be excluded from the returned data.frame
`QC.filter`	a boolean value indicating if the quality control filter should be applied over the data

Details

This function is a wrapper over melt and cast from the reshape package of Hadley Wickham.

The id variables are selected by default. You can use summary.cell.data to see which variables are used as defaults for id.vars. The measured variables can be specified with select and exclude, or with measure.vars.

The casting formula has the following format: x_variable + x_2 ~ y_variable + y_2 ~ z_variable ~ ... | list_variable + ... The order of the variables makes a difference. The first varies slowest, and the last fastest. There are a couple of special variables: '...' represents all other variables not used in the formula and '.' represents no variable, so you can do formula=var1 ~ .

If the combination of variables you supply does not uniquely identify one row in the original data set, you will need to supply an aggregating function, fun.aggregate. This function should take a vector of numbers and return a summary statistic(s). It must return the same number of arguments regardless of the length of the input vector. If it returns multiple value you can use result_variable to control where they appear. By default they will appear as the last column variable.

The margins argument should be passed a vector of variable names, eg. c('pos','t.frame'). It will silently drop any variables that can not be margined over. You can also use 'grand_col' and 'grand_row' to get grand row and column margins respectively.

Value

a reshaped data.frame

Author(s)

Alan Bush

See Also

aggregate

Examples

```
#load example dataset
data(ACL394)

#rehape position 1 in pos + cellID ~ variable + t.frame for f.tot.y variable
reshape(X,select="f.tot.y",subset=pos==1)

#redefining the formula, reshape against time in minutes
X<-transform(X,time.min=10+t.frame*15) #calculating the time of each t.frame
reshape(X,pos+cellID~variable+time.min,select="f.tot.y",subset=pos==1&t.frame<10)
```

`select.cells` *Select Subset of Cells*

Description

Selects a subset of cells that satisfy the specified conditions.

Usage

```
select.cells(X, subset = TRUE, n.tot.subset=NULL ,QC.filter=TRUE)
```

Arguments

`X`	cell.data object
`subset`	a boolean vector of length equal to the number of rows of the dataset, or a conditional statement using the dataset¥s variable, that specifies which registers should be included
`n.tot.subset`	a conditional statement usually involving n.tot, to filter the cells by the total number of frames in which they appear.
`QC.filter`	a boolean value indicating if the QC.filter should be applied over the data

Details

select a group of cells be a criteria specified in `subset`. After the first subset is applied the number of frames in which a selected cell appears (n.tot) is calculated and an additional filter (`n.tot.subset`) is applied. This can be useful to select cells that satisfy the specified `subset` filter in all the time frames, or a fraction of them.

You can do union, intersection and difference of these sets.

Value

Returns a vector of the selected cells 'ucid'. The ucid (or 'unique cell id') is defined as pos*100000+cellID. Because the returned value is an integer vector all the set operations may be applied directly over subsets of cells selected by select.cells. The returned vector should be assigned to a variable for further usage.

Author(s)

Alan Bush

See Also

`intersect`,`union`,`setdiff`

Examples

```
#load example dataset
data(ACL394)

#select cells that have f.tot.y>1e7 in at least one t.frame
c1<-select.cells(X,f.tot.y>1e7)
cplot(X,f.tot.y~t.frame,color="gray",size=0.5) +  #plotting the cells
  clayer(X,f.tot.y~t.frame,color=ucid,geom="line",subset=ucid%in%c1)

#select cells that have f.tot.y<6e5 in all t.frames
c1<-select.cells(X,f.tot.y<6e5,n.tot.subset=n.tot==14)
cplot(X,f.tot.y~t.frame,color="gray",size=0.5) +  #plotting the cells
  clayer(X,f.tot.y~t.frame,color=ucid,geom="line",subset=ucid%in%c1)
```

---

select.vars *Select Variables*

Description

Selects a group of variable names from the dataset.

Usage

```
select.vars(X,select="all",exclude=NULL)
```

Arguments

X	cell.data object
select	character vector defining variables names, keywords or wildcard patters to be included in the returned vector
exclude	character vector defining variables names, keywords or wildcard patters to be excluded from the returned vector

Details

Selects a group of variables. If you only use the first argument it returns 'all' the columns of the dataset.

Value

A character vector with variable names.

Author(s)

Alan Bush

See Also

names

Examples

```
#load example dataset
data(ACL394)

#select all variables
select.vars(X)

#select morphological variables
select.vars(X,"morpho")

#select variables of the YFP channel
select.vars(X,"*.y")

#select id vars, area vars and f.tot.y
select.vars(X,c("id.vars","a.*","f.tot.y"))

#select id vars, area vars and f.tot.y, exlude bg variables
select.vars(X,c("id.vars","a.*","f.tot.y"),exclude="*bg*")
```

---

subset *Subset a Cell Data Objects*

Description

Returns subset of the cell.data object which meet conditions

Usage

```
## S3 method for class 'cell.data'
subset(x,subset=TRUE,select="all",exclude=NULL,QC.filter=FALSE,...)

remove.vars(X,select,exclude=NULL)
```

Arguments

x	cell.data object
X	cell.data object
subset	a boolean vector of length equal to the number of rows of the dataset, or a conditional statement using the dataset¥s variable, that specifies which registers should be included
select	character vector defining variables names to be included in the returned cell.data
exclude	character vector defining variables names to be excluded from the returned cell.data
QC.filter	a boolean value indicating if the quality control filter should be applied over the data before creating the new cell.data object
...	further arguments passed to or used by methods

Details

`subset` is a generic function. This version applies to cell.data objects. `subset` is a close function, meaning it returns an object of the same class as its first argument, in this case a cell.data object. Subsetting is useful to divide a large experiment into smaller dataset that are more easily analyzed. It can also be used to reduce the memory space a cell.data object occupies, for example eliminating the QC filtered registers (`X<-subset(X,QC.filter=TRUE)`) or eliminating unused variables (`X<-subset(X,exclude=c("morpho","f.bg.y","f.*.c"))`)

The bracket (`Extract`) notation can also be used `Y<-X[pos==1]`

`remove.vars` is a wrapper over subset, it eliminates the specified variables.

A record of the subset history of the object is kept. Use `summary.cell.data` to see it.

Value

a subset cell.data object

Author(s)

Alan Bush

See Also

`subset`, `summary.cell.data`

Examples

```
#load example dataset
data(ACL394)

#subset the cell.data by pos
X1<-subset(X,pos==1)
X1<-X[pos==1]

#subset by t.frame and select variables
#note the use of keywords and pattern matching to select the variables
X.t13<-X[t.frame==13,c("morpho","*.y","f.tot.c")]
summary(X.t13) #take a look at the new cell.data object

#eliminate registers that didn't pass the QC filter
X<-subset(X,QC.filter=TRUE)
```

---

summary *Cell Data Object Summary*

Description

Returns a summary of the cell.data object content.

Usage

```
## S3 method for class 'cell.data'
summary(object,...)
```

Arguments

object	cell.data object
...	further arguments passed to or used by methods

Details

Returns a description of the cell.data object, including from where and when it was loaded, the number of positions and time frames and information about the default, transformed and merged variables. It also returns a history of the QC filters and subsets applied.

The function returns a list of class summary.cell.data that is printed by print.summary.cell.data.

Value

a list of class summary.cell.data

Author(s)

Alan Bush

See Also

summary

Examples

```
#load example dataset
data(ACL394)

#see the object summary
summary(X)

#assign the object summary
X.sum<-summary(X)
names(X.sum)
```

---

transform *Transform a Cell Data Object*

Description

Transforms a cell.data object adding new variables

Usage

```
## S3 method for class 'cell.data'
transform(`_data`,...,QC.filter=TRUE)

transform.by(`_data`,.by,...)

## S3 method for class 'cell.data'
transform.by(`_data`,.by,...,QC.filter=TRUE)

## S3 method for class 'data.frame'
transform.by(`_data`,.by,...,subset=NULL)

## Default S3 method:
transform.by(`_data`,.by,...,subset=NULL)
```

Arguments

`_data`	cell.data object or data.frame to transform
`.by`	variables to split data frame by, as quoted variable
`...`	new variable definition in the form `tag=value`
`QC.filter`	a boolean value indicating if the quality control filter should be applied over the data
`subset`	logical expression indicating elements or rows to keep: missing values are taken as false.

Details

Read the transform vignette for a tutorial on the use of these functions > vignette("transform")

`transform.cell.data` is the implementation of the generic function `transform` to cell.data objects. It creates the new variables based on the `...` argument; a tagged vector expressions, which are evaluated in the dataset.

`transform.by` is a generic function. Before transforming the dataset, the function splits it by the variables specified in the `.by` argument. This argument should be a quoted list of variables, that can be easily created with the `quoted` function, for example `.(pos,t.frame)`. This can be useful to do group-wise normalizations.

The transformed variables are summarized in the output of `summary.cell.data`.

Value

for transform(.by).cell.data a transformed cell.data object

for transform.by.data.frame a transformed data.frame

Author(s)

Alan Bush

See Also

`transform`

Examples

```
#load example dataset
data(ACL394filtered)

#creating a new variable
X<-transform(X,f.total.y=f.tot.y-a.tot*f.local.bg.y)

#create a new variable normalizing by position
X<-transform.by(X,.(pos),norm.f.total.y=f.total.y/mean(f.total.y))

#create a new delta variable in sigle cells
X<-transform.by(X,.(pos,cellID),delta.f.total.y=f.total.y-f.total.y[t.frame==0])
```

`transform.cell.image.rd`

Transform Cell Image

Description

funcionts that transforms a cell image object before plotting

Usage

```
cnormalize(X=NULL,normalize.group=c("channel"),...)
```

Arguments

`X`	cell.image object to transform
`normalize.group`	character vector indicating with variables should be used to group the images for normalization
`...`	further arguments for methods

Details

These functions are called from `cimage` to transform the images in some way before plotting.

`cnormalize` applies a normalization to the images that to enhance contrast. The normalization groups (defined by `normalize.group`) are applied the same normalization, so the intensities can be compared within a group.

if `X` is NULL, the funcion returns a character indicating with variables of the dataset it requires.

Value

The transformed cell.image object

Author(s)

Alan Bush

See Also

cimage

Examples

```
#load example dataset
data(ACL394)

#correcting the path to the images
#normally you won't need to do this
X$images$path<-factor(system.file('img', package='Rcell'))

#select N=3 cells images from each pos (group),
#from the first t.frame and pos 1,8,15,22,29.
ci<-get.cell.image(X,subset=match(pos,c(1,8,15,22,29),nomatch=0)>0&t.frame==11,
group=.(pos),N=3,channel=c('BF','YFP'))

#display a cell image without normalization
if(interactive()) display(tile(combine(ci)))

ci<-cnormalize(ci) #apply normalization
if(interactive()) display(tile(combine(ci))) #display again
```

---

update.n.tot *Calculate Total Number of Frames for Each Cell*

Description

updates n.tot, the total amounts of frames in which a given cell appears

Usage

```
update.n.tot(object, QC.filter = TRUE,...)
```

Arguments

object	cell.data object
QC.filter	a boolean value indicating if the quality control filter should be applied
...	futher arguments for methods

Value

returns a cell.data object, with updated values for n.tot

Author(s)

Alan Bush

See Also

load.cellID.data,select.cells

Examples

```
#load example dataset
data(ACL394)

#update n.tot variable
X<-update.n.tot(X)

#this command is equivalent to
X<-transform.by(X,.(ucid), n.tot=length(t.frame))
```

`with` *Evaluates an Expression in a Cell Data Object.*

Description

Evaluate an R expression in an environment constructed from the cell.data object.

Usage

```
  ## S3 method for class 'cell.data'
with(data,expr,subset=TRUE,select=NULL,exclude=NULL,QC.filter=TRUE,...)
```

Arguments

`data`	cell.data object
`expr`	expression to evaluate
`...`	arguments to be passed to future methods
`subset`	a boolean vector of length equal to the number of rows of the dataset, or a conditional statement using the dataset¥s variable, that specifies which registers should be included
`select`	character vector defining variables names to be included
`exclude`	character vector defining variables names to be excluded
`QC.filter`	a boolean value indicating if the quality control filter should be applied over the data

Details

`with` is a generic function. The version for cell.data objects is a wrapper over the version for data.frame, calling `as.data.frame.cell.data` with the specified arguments.

Value

The value of the evaluated `expr`

Author(s)

Alan Bush

See Also

with

Examples

```
#load example dataset
data(ACL394)

#calculate the mean f.tot.y from pos 2
with(X,mean(f.tot.y[pos==2]))

#use base plotting
with(X,plot(f.tot.y~f.tot.c))
```

---

zoom *Zoom in a ggplot Object*

Description

Sets the plotting region and axis breaks for a ggplot object

Usage

```
zoom(xzoom=c(NA,NA),yzoom=c(NA,NA),nx.breaks=n.breaks,ny.breaks=n.breaks
  ,n.breaks=7)
xzoom(xzoom=c(NA,NA),nx.breaks=7)
yzoom(yzoom=c(NA,NA),ny.breaks=7)
```

Arguments

xzoom	numeric vector of length 2 specifying the range of the x axis
yzoom	numeric vector of length 2 specifying the range of the y axis
nx.breaks	number of breaks for the x axis
ny.breaks	number of breaks for the y axis
n.breaks	number of breaks for both axis, if not specified by nx.breaks or ny.breaks

Details

xzoom and yzoom are convenient functions to specify only one of the limits

Value

a layer to be added to a ggplot object, that specifies the plotting region after the statistical transformations have been done.

Note

A zoom function exists in Hmisc package. Use Rcell::zoom if both package namespaces are loaded.

Author(s)

Alan Bush

See Also

cplot,limits

Examples

```
#load example dataset
data(ACL394)

#zoom in the y axis
cplotmeans(X,f.tot.y~t.frame,color=pos) + zoom(y=c(0,7e6))
```

Index